Good properties of hand-written signatures:

1. Signature is authentic.
2. Signature is unforgeable.
3. Signature is not reusable (it is a part of the document)
4. Signed document is unalterable.
5. Signature cannot be repudiated.

What problems do we want into if we want to achieve all this in digital signatures?
Signatures Scheme

To sign: use a **private signing** algorithm
To verify: use a **public verification** algorithm

In particular:

Alice wants to sign message $m$. She computes the signature of $m$ (let’s call it $y$) and sends the **signed message** $(m,y)$ to Bob.

Bob gets $(m,y)$, runs the verification algorithm on it. The algorithm returns “true” iff $y$ is Alice’s signature of $m$.

**Example with RSA-like setting:**

Alice: $y = m^d \mod n$ \quad sends $m,y$ to Bob
Bob: \quad \text{verifies } m = y^e \mod n \quad \checkmark

How can we do this?
Signatures Scheme

Some public-key cryptosystems can be used for digital signatures, for example RSA, Rabin, and ElGamal:

The basic protocol:

1. Alice encrypts the document with her private key.

2. Alice sends the signed document to Bob.

3. Bob decrypts the document with Alice’s public key.
RSA Signature Scheme

1. Alice chooses secret odd primes $p,q$ and computes $n=pq$.
2. Alice chooses $e_A$ with $\gcd(e_A, \Phi(n)) = 1$.
3. Alice computes $d_A = e_A^{-1} \mod \Phi(n)$.
4. Alice’s signature is $y = m^{d_A} \mod n$.
5. The signed message is $(m,y)$.
6. Bob can verify the signature by calculating $z = y^{e_A} \mod n$. (The signature is valid iff $m=z$).

Potential issues:

- Eve could send $(y_1^{e_A} \mod n, y_1)$ to Bob. Is this a problem?

  appears signed by Alice but message garbage

  \rightarrow no problem
RSA Signature Scheme

1. Alice chooses secret odd primes $p, q$ and computes $n = pq$.
2. Alice chooses $e_A$ with $\gcd(e_A, \Phi(n)) = 1$.
3. Alice computes $d_A = e_A^{-1} \mod \Phi(n)$.
4. Alice’s signature is $y = m^{d_A} \mod n$.
5. The signed message is $(m, y)$.
6. Bob can verify the signature by calculating $z = y^{e_A} \mod n$. (The signature is valid iff $m = z$).

Potential issues:

- Eve could send $(y_1^{e_A} \mod n, y_1)$ to Bob. Is this a problem?
- Bob can reuse the signed message. When would this be a problem? (e.g., if message is "I agree")
Attacks on Signature Schemes

Typical types of attacks for cryptosystems: ciphertext-only, known-plaintext, chosen-plaintext, and chosen-ciphertext.

Typical types of attacks for signature schemes:

- **key-only**: know the public key but have no signed document

- **known-message**: know message + its signature (+ the public key)

- **chosen-message**: choose a message and get it signed
Attacks on Signature Schemes

Additionally, Eve can have different goals:

- **total break**: Eve determines Alice’s signing key/function.

- **selective forgery**: Eve is able (with nonnegligible probability) to create a valid Alice-signature on a message chosen by someone else.

- **existential forgery**: Eve is able to create a valid signature for at least one new message.
Some Breaks for RSA Signatures

We mentioned Eve sending \((y_A^e \mod n, y)\) to Bob. What type of attack is this? *key only* What goal does it achieve? *existential forgery*

If Eve has two signed messages \((m_1, m_1^{d_A} \mod n)\) and \((m_2, m_2^{d_A} \mod n)\), then Eve can create a valid signature on \(m_1 m_2 \mod n\). How?

What type of attack is this? *known message* What goal does it achieve? *existential forgery*

Eve can also do a selective forgery using a chosen message attack. How? *get Alice to sign something*
Blind Signatures

Bob wants to time-stamp his document by Alice, without revealing its content to Alice.

1. Alice chooses secret odd primes $p$, $q$ and computes $n = pq$.
2. Alice chooses $e$ with $\gcd(e, \Phi(n)) = 1$.
3. Alice computes $d = e^{-1} \mod \Phi(n)$.
4. Bob chooses a random integer $k$ (mod $n$) with $\gcd(k, n) = 1$, and computes $t = k^e m \mod n$, where $m$ is the message.
5. Alice signs $t$, by computing $s = t^d \mod n$. She sends $s$ to Bob.
6. Bob computes $k^{-1}s \mod n$. This is the signed message $m^d$.

Why?

Bob gets $s = t^d \mod n = (k^e m)^d \mod n = k^{ed} m^d \mod n = k^{-1} m^d \mod n$

$k^{-1} s \equiv m^d \mod n$  \[ \arrowright \]  k^{-1} s \equiv m^d \mod n, \text{Bob has signature of m}

This protocol is good for Bob but not very good for Alice since she does not know what she is signing!
Insecurity of RSA against Chosen-Ciphertext

Let’s revisit this attack (see earlier slides).

Given a ciphertext $y$, we can choose a ciphertext $\hat{y} \neq y$ such that knowledge of the decryption of $\hat{y}$ allows us to decrypt $y$.

Eve chooses $\hat{y} = ey$ where $y^e = x_0 \mod n$ for some random $x_0$ (e.g. make $x_0 = 2$)

Eve gets Alice to blindly sign $\hat{y}$:

Eve gets $\hat{y}^d = (ey)^d = y^d = x_0^d = x_0 \cdot x$ (e.g. $x^d = y$)

Eve computes $x_0^{-1} \cdot$ signature of $\hat{y} = x$

This decrypts $y$

Moral of the story: Do not sign documents blindly
Combining Signatures with Encryption

If Alice wants to both sign and encrypt a message for Bob:

Either:
Alice signs her message, then encrypts the signed message. I.e. Alice sends $e_{Bob}(m, \text{sig}_{Alice}(m))$, where $e_{Bob}$ is Bob’s (public) encryption function and $\text{sig}_{Alice}$ is Alice’s (private) signing function.

Or:
Alice encrypts the message, then signs the encrypted message. I.e. Alice sends $(e_{Bob}(m), \text{sig}_{Alice}(e_{Bob}(m)))$.

Which way is better?
Hash Functions

Signature schemes: typically only for short messages (for the RSA signature scheme, messages need to be from $\mathbb{Z}_n$).

What to do with longer messages?

Naïve solution:

- Cut up into chunks of $\mathbb{Z}_n$, sign each separately.
- Problems:
  1. Signature as long as the message.
  2. Could rearrange the order of the chunks in the message.
Cryptographic Hash Functions

Using a very fast public cryptographic hash function $h$, we can create a message digest (or hash) of a specified size (e.g. 160 bits is popular).

What does Alice do?

Alice wants to sign message $m$:
1. computes $h(m)$
2. signs $h(m)$: $\text{sign}_A(h(m))$
3. sends $(m, \text{sign}_A(h(m)))$ to Bob

How does Bob verify the signature?

1. computes $h(m)$
2. verify signature $\text{sign}_A(h(m))$ against $h(m)$
Other uses of cryptographic hash functions:

- Data integrity

  hash func. should produce different results if the message gets corrupted, e.g. during transmission

- Time stamping a message while keeping the message secret

  how to convince people that a document was not altered after its creation
  
  \[ \rightarrow \text{compute } h(m), \text{ publish it} \]
  \[ \text{then, if unaltered, } h(m) \text{ would change once made public, then we can verify } h(m) \text{ is the same as the published hash} \]
Signed Hash Attacks

We have to make sure that $h$ satisfies certain properties, so that we don't weaken the security of the signature scheme.

**Attack 1:**
Eve finds two messages $m_1 \neq m_2$ such that $h(m_1) = h(m_2)$. Eve gives $m_1$ to Alice, and persuades her to sign $h(m_1)$, obtaining $y$. Then $(m_2, y)$ is a valid signed message.

To prevent this attack, we require that $h$ is collision resistant (or strongly collision-free), i.e., it is computationally infeasible to find $m_1 \neq m_2$ such that $h(m_1) = h(m_2)$. 
Signed Hash Attacks

We have to make sure that h satisfies certain properties, so that we don't weaken the security of the signature scheme.

Attack 2:
Suppose Eve can forge signatures on random message digests. For example, in RSA, z is the signature of $z^e_A$. If Eve can find m such that $z^e_A = h(m)$, then (m, z) is a valid signed message.

To prevent this attack, we require that h is oneway (a.k.a. preimage resistant), i.e., given y, it is computationally infeasible to find m such that $h(m) = y$. 
Size of Hashes

The birthday paradox: what is the chance of having 2 or more people with the same birthday in a class of 23 people?

\[
\text{prob. of no collisions: } \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \left(1 - \frac{3}{365}\right) \cdot \left(1 - \frac{4}{365}\right) \cdots \cdot \left(1 - \frac{22}{365}\right) \approx 0.5
\]

What does it have to do with hashing?

The birthday paradox in general:

Roughly \( \sqrt{n} \) elements randomly placed into a table of size \( n \) produce a collision with 50% chance.

Moral of the story: keep this in mind when designing your hash function.
Creating Hash Functions

Theoretically appealing option: creating hash functions from oneway functions, e.g. the Discrete Log (coming soon)

In practice (since the above is too slow): There are several professional strength hash functions available. E.g., MD4, MD5, and SHA-1 (of similar structure as the MDs): $2^{64}$ bits hashed into 160 bits. In 2001, NSA published SHA-2, four cryptographic hash functions: $2^{128}$ bits into 224 to 512 bits. In October 2012, SHA-3 competition winner Keccak: $2^{128?}$ bits into 224 to 512 bits.

Actively researched: MD4, MD5: known weaknesses, SHA-1: theoretical weakness. SHA-3 meant as alternative to SHA-2.
In 1991, NIST proposed DSA for use in their Digital Signature Standard (DSS). It was adopted in 1994.

There were several criticisms against DSA:
1. DSA cannot be used for encryption or key distribution.
2. DSA was developed by the NSA, and there may be a trapdoor in the algorithm.
3. DSA is slower than RSA.
4. RSA is the de facto standard.
5. The DSA selection process was not public.
6. The key size (512 bits) is too small. In response to this criticism, NIST made the key size variable, from 512 to 1024 bits.
Discrete Log

DSA gets its security from the difficulty of computing the discrete log.

**Discrete Log problem:**
Fix a prime p. Let $\alpha$ and $\beta$ be nonnegative integers mod p, the goal is to find the smallest natural number $x$ such that $\beta \equiv \alpha^x \pmod{p}$. The number $x$ is denoted by $L_\alpha(\beta)$: the discrete log of $\beta$ with respect to $\alpha$.

Often, $\alpha$ is taken to be a primitive root mod p. $\alpha$ is a **primitive root mod p** if and only if \{ $\alpha^i \pmod{p} \mid 0 \leq i \leq p-2$ \} = {1, 2, ..., p-1}.

For example:
- 3 is a primitive root mod 7
- 2 is a primitive root mod 13, but 3 is not
If $\alpha$ is a primitive root mod $p$, then $L_\alpha(\beta)$ exists for all $\beta \neq 0 \pmod{p}$.

If $\alpha$ is not a primitive root mod $p$, then $L_\alpha(\beta)$ may not exist. For example, the equation $3^x \equiv 2 \pmod{13}$ does not have a solution, so $L_3(2)$ does not exist.

There are $\Phi(p-1)$ primitive roots mod $p$.

Like factoring, the discrete logarithm problem is probably difficult.

Recall: the ElGamal public-key cryptosystem is based on discrete log.
ElGamal Signature Scheme

Alice (beforehand):
1. Select a large prime \( p \), and a primitive root \( \alpha \).
2. Select \( a, 1 \leq a \leq p-2 \).
3. Compute \( \beta = \alpha^a \mod p \).
4. Publish \( p, \alpha, \beta \).

Alice (to sign a message \( m \)):
1. Select random \( k \) such that \( \gcd(k, p-1) = 1 \).
2. Compute \( r = \alpha^k \mod p \), and \( s = k^{-1}(m-ar) \mod (p-1) \).
3. Send \((m, r, s)\) to Bob.

Bob (to verify):
1. Compute \( v_1 = \beta^{rs} \mod p \), and \( v_2 = \alpha^m \mod p \).
2. Signature valid iff \( v_1 = v_2 \).
ElGamal Signature Scheme - remarks

How many signatures per message?

- as many as coprimes of \( p-1 \) (a select \( k \))
- \( \varphi(p-1) \)

Notice: no need to compute the logarithm...

Eve’s options to break the signature scheme:

- find \( a \) - we believe hard
- to get a specific message signed, generate random \( r \) and compute corresponding \( s \) to convince Bob to accept the signature

Then, she wants: \( v_1 = v_2 \) i.e.

\[
\begin{align*}
\beta^r \cdot r^s & \equiv \alpha^{-1} \pmod{p} \\
\Rightarrow r^s & \equiv \alpha^{-1} (p-1)^{-1} \pmod{p}
\end{align*}
\]

Another option:

- if Alice chooses the same \( k \) for two messages

Then \( (m_1, r_1, s_1) \) and \( (m_2, r_2, s_2) \) are the two signed messages

Then

\[
\begin{align*}
\begin{array}{l}
s_1 \equiv k^3 (m_1 - ar) \\
s_2 \equiv k^3 (m_2 - ar)
\end{array}
\end{align*}
\]

\( \Rightarrow \ ar \equiv m_1 - s_1, k \equiv m_2 - s_2 \ k \Rightarrow m_1 - m_2 \equiv (s_2 - s_1) k \pmod{p-1}

Once have \( k \), similarly compute \( a \) from

\( \begin{align*}
\begin{array}{l}
an \equiv m_1 - s_1, k
\end{array}
\end{align*}
\)

Usually not too many, try them

\( \begin{align*}
\begin{array}{l}
\text{Solve the congruence to get } k \\
\text{(possibly several solutions, } \gcd(s_2 - s_1, p-1))
\end{array}
\end{align*}
\)
Alice's init:
1. Find 160-bit prime q, and find a prime (512+)-bit p s.t. q | p-1.
2. Let g be a primitive root mod p, let $\alpha = g^{(p-1)/q} \mod p$.
3. Choose $a$, $1 \leq a < q-1$, calculate $\beta = \alpha^a \mod p$.
4. Publish $(p, q, \alpha, \beta)$.

Alice, to sign a message m:
1. Select random (secret) k, $0 < k < q-1$.
2. Compute $r = (\alpha^k \mod p) \mod q$, and $s = k^{-1}(m + ar) \mod q$.
3. Send $(m, r, s)$ to Bob.

Bob, to verify:
1. Compute $u_1 = s^{-1}m \mod q$, and $u_2 = s^{-1}r \mod q$.
2. Compute $v = (\alpha^{u_1} \beta^{u_2} \mod p) \mod q$.
3. Accept the signature iff $v=r$. 